

jMax Phoenix: le jMax nouveau est arrivé

[The new jMax has come]

Maurizio DE CECCO

jMax Phoenix project

<http://www.jmax-phoenix.org>
maurizio@jmax-phoenix.org

Francois DECHELLE

jMax Phoenix project

<http://www.jmax-phoenix.org>
francois@jmax-phoenix.org

Enzo MAGGI

jMax Phoenix project

<http://www.jmax-phoenix.org>
enzo@jmax-phoenix.org

Abstract

The reports of the jMax death have been greatly exaggerated. Free software never dies, it just sleeps for some time. Almost nine years after the release of the project under a free license, and six years after the end of the developments by the institution that created it, some of the original project developers decided to revive it from its ashes: jMax Phoenix was born.

Keywords

Max, jMax, Patcher.

1 The Max Language

Max, invented by Miller Puckette at Ircam, is essentially a visual programming language. By reusing the analog modular synthesizer metaphor, it allows to build *patches* by connecting modules together with *patch cords*. Via connections, control messages or signals are sent between the modules.

These modules generally represent processing or data units, or system input/outputs, or interactive controllers. The simplicity and directness of the graphical metaphor account for the Max language success.

2 The Max History

The history of Max go back to the Patcher editor[2] developed by Miller Puckette for controlling the Ircam 4X machine[1]. The Patcher was licensed to Opcode where it was re-engineered by David Zicarelli as the product known as Max Opcode.

In 1989, a new version of the Patcher editor, known as Max/ISPW[4] was developed in Ircam within the Ircam Signal Processing Workstation project[3,4]. Max/ISPW extended the initial paradigm adding real-time signal processing modules and connections.

In 1994 Ircam started the development of a purely software environment to substitute the expensive custom hardware used in the past. The first result of this project was Max/FTS[5,6], a version of Max/ISPW running on Silicon Graphics workstations, using X/Motif for its user interface, and a processing engine written in C.

In 1996 Miller Puckette, at that time already at the University of San Diego (CA), published the first version of Pure Data (also known as PD), a new version of Max that provided signal processing features and a Tcl/Tk based user interface. In 1997 David Zicarelli included signal processing in the commercial Max, releasing a new version of MAX called Max/MSP.

In 1996 Francois Dechelle and Maurizio De Cecco, later joined by Enzo Maggi and Norbert Schnell, started in Ircam the development of jMax[7,8,9,10,11], a new generation of the Max systems. jMax included a complete re-implementation of the computational engine, providing a simpler and portable API for objects, and a Java based user interface. jMax ran on multi-processor Silicon Graphics machines and was used for several high-end productions in Ircam. In the summer of 1999 Ircam published the source code under a GPL-like license.

Later in the same year, Maurizio De Cecco and Enzo Maggi left Ircam to follow new opportunities in the free software world. The development of jMax seems to stop in 2002, even if the project was shutdown more or less formally only in 2004.

In the summer 2008 Maurizio De Cecco successfully involved his old colleagues Francois Dechelle and Enzo Maggi in restarting jMax and to transform it in a community project.

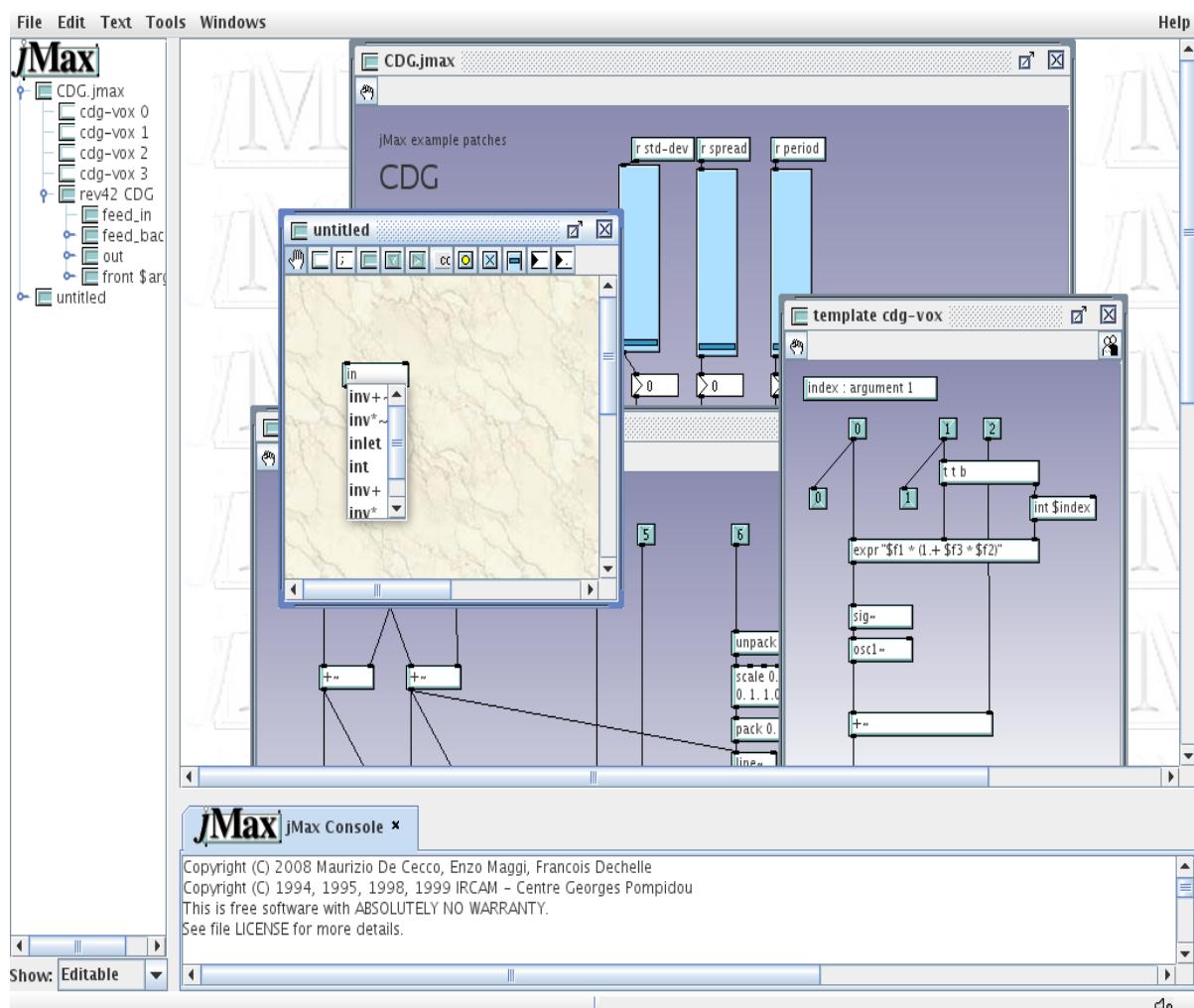
3 Why jMax Phoenix ?

There a number of good reasons to bring back jMax to life: jMax was an advanced project with respect to its times; jMax was written for high end hardware in mind: at least two high performance processors were required to run it correctly. Today any modern desktop or laptop configuration satisfy

this requirement; the run-time architecture of jMax fit well in modern multi-core architectures and its modular architecture gives a lot of headroom for future evolutions.

The project was also very in advance in its use of Java as a portable UI environment ; in 1996 the JVM performances were not optimal, and the Swing UI toolkit was just a first buggy approximation of what we have today.

No free implementations of Java were available, thus reducing the interest of the free software community in jMax. Today, Java and Swing implementations are very complete and have excellent performances. The recent release of the Sun's JVM under a GPL license provides a free java implementation to the open source community.



Screen shot 1: The new JMax user interface (ide style)

Even if for a number of reasons jMax was at a certain point of its development on a "dead branch" and then abandoned, jMax has never fallen out completely of public interest. The Max language is still alive and kicking, as the success of MAX/MSP and Pure Data proves.

In synthesis, what were at that time the limits of jMax, are today his strengths: it can use multiple processors, it implements a very reach and powerful graphic environment, for which completely free implementation exists, and has an architecture defined for the future.

Least but not last: jMax is the finest and funniest project the authors had the occasion of working on; the original developers of jMax are still around, with enough competences on the code to kick-start the developments of a new generation jMax.

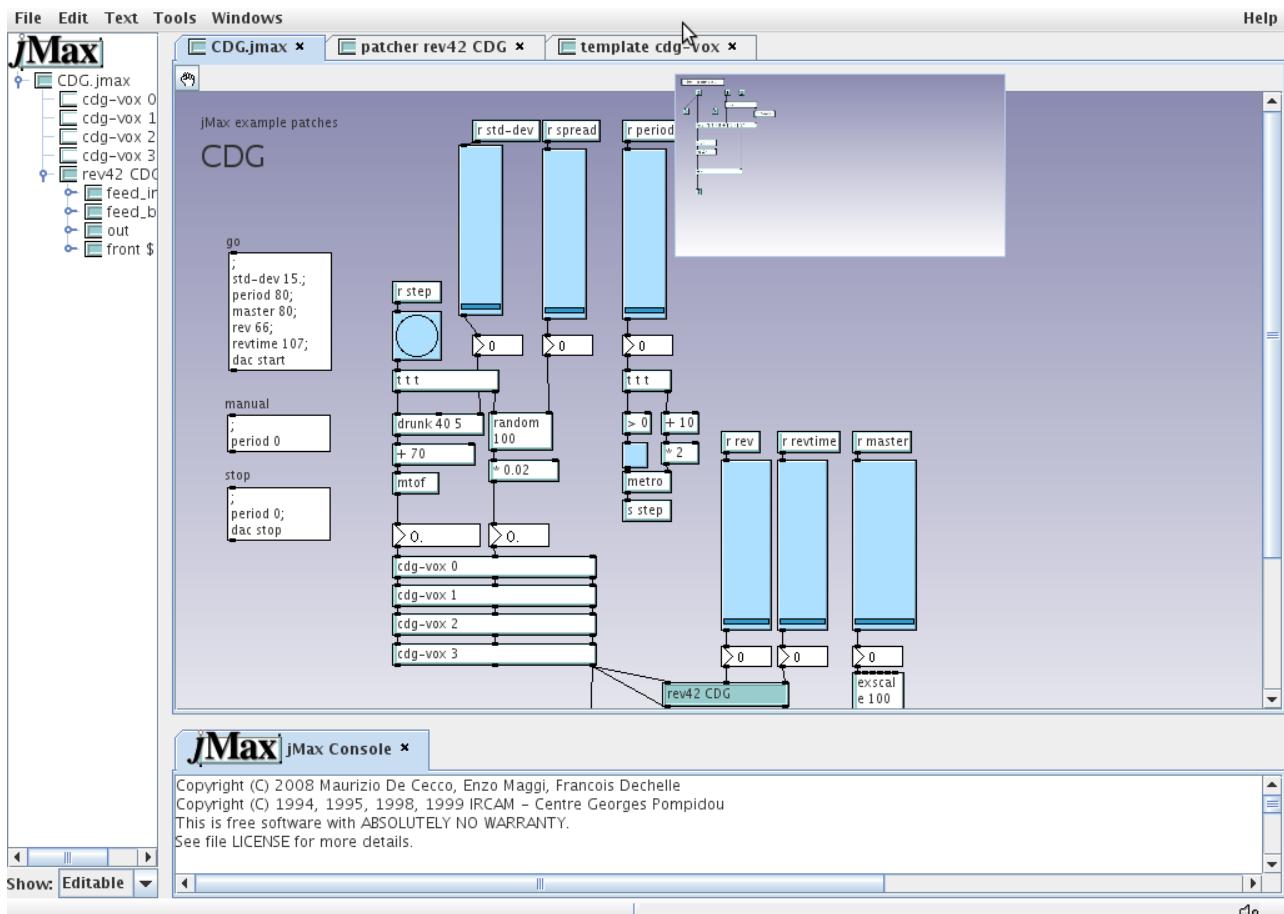
4 User Interface

The jMax Phoenix initial focus is to "refresh" and extend the jMax user interface, in order to renew the traditional patcher application paradigm and provide a more productive and attractive UI.

4.1 The MAX User Interface Paradigm

The classic Max user interface paradigm is based on the UI style of the 80s: a lot of small unrelated windows composing the application user interface. The UI focuses on the console as the main interaction place, and every tool and editor is opened in a new window.

While this organization is simple and effective for simple applications and a limited number of editors/tools, it is not appropriate (and even problematic) when an application is composed by tens or hundreds of patches, and in presence of tens of different tools and editors.



Screen shot 2: The jMax tabbed user interface

Essentially, the paradigm do not scale with the complexity of the application, and with the richness of the development and execution environments.

4.2 The jMax Phoenix User Interface design

The new user interface for jMax should give a rich set of choices to the final users, leaving him the freedom to choose his way of working. So, the design of the new user interface provides a set of rich, user- and developer- extensible mechanisms and a set of pre-built choices for an easy start.

jMax Phoenix user interface implementation separates the editors and tools from the container ; a number of different containers allows the implementation of different, configurable user interface styles.

The first container implements the classic Max user interface: each editor and tool opens in an independent window, and the set of opened windows represents the user interface of the running application. This behavior is maintained for compatibility with the standard Max behavior, but it may also be appropriate for some application, especially as an execution environment.

The classic environment can be declined in a family of environments: the default one keeps the Max console at the center of the user interaction. Other configurations are possible, for example a tool-less configuration that opens only application windows at start up, or an user interface where the project browser takes the place of the console.

A more powerful container provides a configurable docking framework, allowing the arbitrary organization of sub-panel in a unique container window. Each panel can contain a single jMax component, or a complex container using a tabbed or a virtual desktop organization.

The configuration of the framework is defined by a script, that allows for a very easy definition of custom layouts. The composition of sub-panels and container allows for a large choice of user interfaces, going from the classic MAX behavior within a single window to the modern IDEs style .

4.2.1 The jMax IDE Style

The IDE style (screen shot 1) takes over the principles from classic IDEs like Eclipse and Netbeans. On the left panel, a browser allows full navigation in the application. The tools, like the console, the find and control panels and the others are grouped in the tool panel on the bottom and organized using tabs. The center area is organized as a virtual desktop, containing the individual patchers in independent windows.

An improved status bar allows a quick check of essential parameters, like DSP status, sampling frequency, latency and so on.

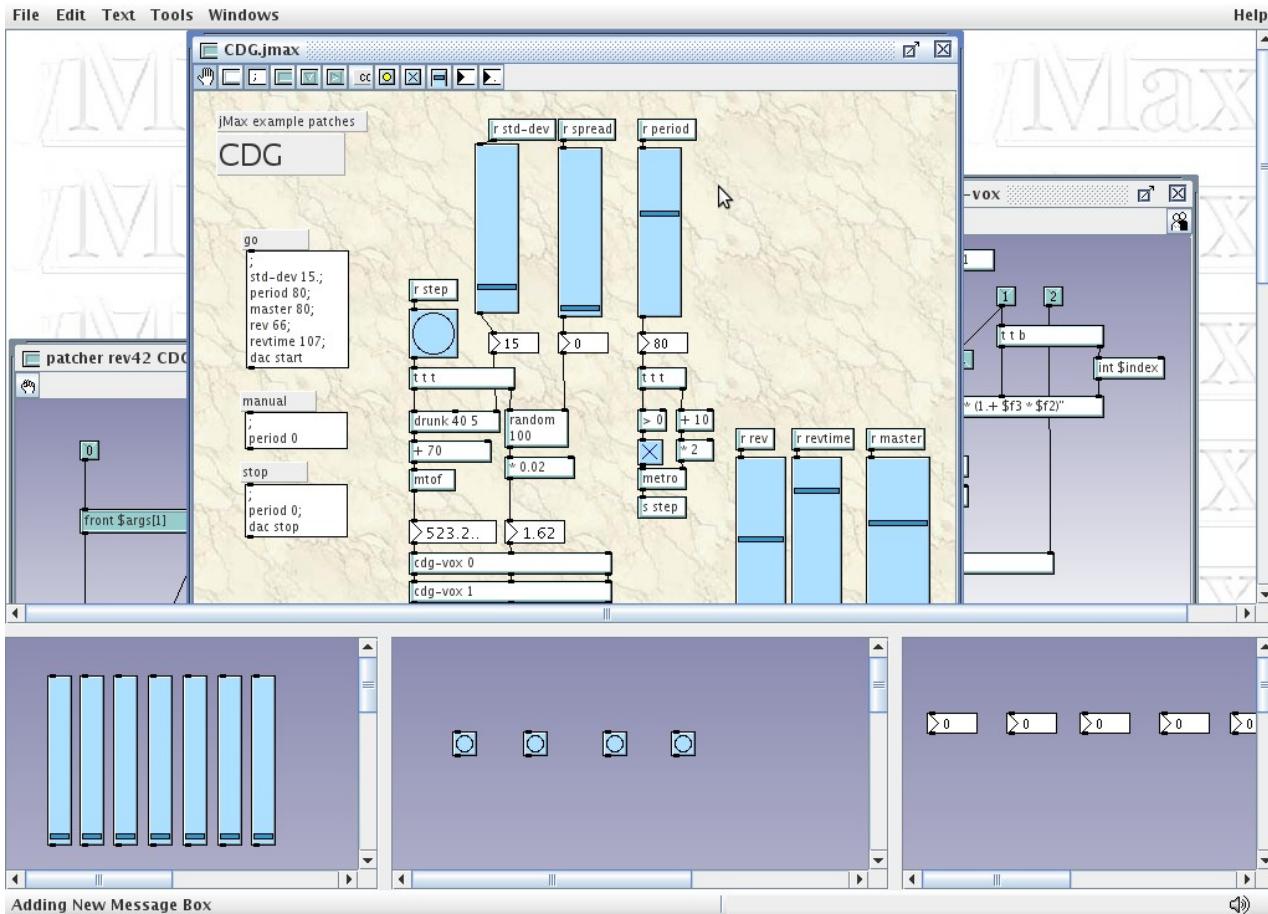
This configuration can be seen as a development environment for applications to be used in the classic or in a custom configuration, providing a faster and clearer access to the available tools.

As a variation, the tabbed IDE style (screen shot 2) keeps the same configuration, but organizes the different panels in a tabbed panel instead of a virtual desktop area. This allows an optimal use of the available screen space, making it especially adapted to small laptop screen, and allowing an easier navigation. A graphic tool-tip allows the inspection of a tab content without changing the open tab.

4.2.2 Custom styles

A custom styles can implement an application dependent interface. An example is shown in screen-shot 3: three fixed windows are opened in the bottom area of the jMax frame, and a central virtual desktop area allows free navigation in the patcher structure. This allows to keep critical application controllers and parameters always accessible, while preserving the flexibility of the standard Max patcher browsing.

This specific user interface is built with just twelve lines of scripting; built-in user interface styles used between five and ten lines of configuration script. Future evolutions may include the interactive generation of configuration script directly from the user interface.



Screen shot 3: A custom, application dependent user interface

4.3 The Patcher Editor

Some work has been spent to refresh the look of the patcher editor and to provide new functionalities. The menus have been rationalized, and a new look using the capabilities of the Java 2D graphic framework has been added, in particular gradients and textures.

Key bindings are being cleaned up and aligned with the modern UI standards. The editor now support the auto-completion of object names; and a meaningful and effective contextual menu.

4.4 The Project Browser

The project browser allows the navigation in the loaded patches. The browser can show all the objects in a patcher, only the objects that have an associated editor or only the patchers, according to the filtering rule. Ctrl-Double-Click on objects open the associated editor if any.

The project browser will be soon complemented by a generic property panel, able to give detailed informations on edited objects, patches and other resources.

5 The FTS Engine

This section is more speculative and discusses the evolutions of the jMax Phoenix computing engine (FTS) for the next year.

5.1 The Max Computation Model

While jMax already make use of multiple threads and processes (notably for the user interface and for direct to/from disk audio streaming), FTS, the computational engine behind the jMax interface, uses a traditional single threaded poll-for-input, compute, output loop.

The loop handles real-time inputs like audio and MIDI, the control related inputs from the user interfaces, the DSP computation and the control computations, everything in the same thread.

This approach implies a well known number of problems. First of all, jMax has never been an hard real time system: control computations (like the computation activated by moving a slider too fast) or system computations (like loading a patch) can slow down DSP computation, and cause severe under-run of the audio stream (and the corresponding output noise).

Second, FTS cannot use more than one CPU core; at the time jMax was originally developed, machines with more than 2 processors were horribly expensive, and the problem was relatively minor. Today, a four core machines costs less than one thousands Euro; also, single core processing power is somehow saturated. Scaling up jMax implies being able to use as many cores as needed.

Finally, a single threaded architecture does not match very well with the callback based audio system like jack. Integrating with jack, and being a good, real-time citizen of the jack ecosystem is a priority to allow a professional use of jMax Phoenix.

5.2 A New Architecture For FTS.

The new architecture of FTS will put multi threading at the core of the execution model and of the MAX language execution model: the new FTS engine will have a thread for each event source, normally waiting for events. Standard synchronization primitives will be used to maintain the global consistency of computation.

Threads will not have the same priority: the DSP thread (or threads) will run at the special real time priority to avoid sound stream interruptions. Different control streams will get different priorities, assuring that no amount of user interface interaction could disrupt the sound output, and that control computation timed by the audio computation will anyway be executed in real time.

This model will also allows FTS to fit nicely in a callback based audio subsystem, like jack.

The implementation of the communication between control computation and DSP computation in a multi-threaded environment has

some critical aspect. Currently, the coherency of the modifications of the state of the DSP engine is guaranteed by the sequential computational model.

Once the control computation is asynchronous with respect to the DSP computation, coherency may become a tricky problem: for example in the case where some coefficients for a specific filter are computed by different parts of the control patch; the current model guarantees that the whole control computation is seen atomically by the DSP computation, i.e. either all the coefficients are changed, or no coefficients are changed.

In a multi-threaded model the DSP thread may kick in before the control computation is completed, getting a part of the new coefficients values and a part of the old ones. The result is that the filter will run with a set of incoherent coefficients, that may cause any possible wild result, like oscillation and filter instability.

There are different options for solving this problem: the most elegant and complete one, and also the most complex, consists in integrating a transactional handling of state changes in the FTS engine itself. A simpler solution is to provide some primitives to handle synchronization in the Max language itself. The jMax Phoenix project will experiment a number of solutions in the next year, testing first the adequacy of the simplest one.

5.3 Parallelism And The Max Language

Using a multi threaded engine also offers the opportunity of supporting parallelism directly in the Max control language, by providing a simple set of objects that will allow the execution of a part of a patch in a new thread.

These objects will implement the traditional parallel operations: *fork*, to start a new thread, *join*, to wait for a thread completion, and *mutex*, to guarantee mutual exclusion on a part of a patch. In general, parallel programming is not easy, and parallel programming in jMax will not be easier; but the added functionalities can be a real bonus in many cases.

5.4 The New DSP Engine

A refresh of the jMax DSP engine is long overdue, with three main objectives: giving more

flexibility in DSP computation, exploiting the power of multi-core architectures, taking advantage of the many opportunities for optimizing the DSP graph.

The main goal is offering more flexibility to DSP programmers by providing a richer computation engine, that will offer the possibility to specify different sample rate and different tick size and rate for different objects.

This would allows the implementations of patches using different time resolution and sampling rate for different purposes. For example, a high resolution high overhead 192Khz/256 sample per tick audio stream can implements the main application audio stream, a lower overhead 48Khz/256 sample per tick stream may be used for audio analysis, and a stream 48Khz/ 2048 sample per ticks stream may bring information in the frequency domain; a 750 Hz/1 sample per tick stream can be used to implements synchronous data flow control, instead of the traditional event driver control.

The second aim is to add a layer of DSP graph optimization to the FTS DSP compiler, able to perform standard optimizations like dead code elimination and constant propagation, and FTS specific optimizations, like operations aggregation.

This optimization will aggregate basic operations defined in the patch in more complex group (like multiply-add) to better exploit the available hardware. Past studies shown that this kind of optimization can easily provide a performance gain between 10 and 20% in a large scale application.

The final and most complex objective is to support a multi-threaded multi-core DSP engine; the task implies some pretty complex graph analysis and optimization problems, as minimizing the latency added by the multiple threads, minimizing the communications between the threads and avoiding as much as possible synchronization points between the threads.

6 Conclusion

The jMax project is alive again; this time it is not funded by any big institution; it is a community project and its future depends on the interest of the developers.

The core team's first work is focused on the basic improvements this paper describes. The Linux/jack environment is the reference environment. Versions for other platforms will be available only if other developers decide to contribute them.

The community is open; the core architecture coherency will be managed by the project leader, while contributed extensions and packages will be self managed. The project subversion repository, web site and mailing lists are open to third party contributions.

Since the project will not dispose of large resources, its priorities for the code evolution will be the stability and the visible added value for the end user. The project can be reached at www.jmax-phoenix.org, email contact@jmax-phoenix.org.

7 Acknowledgments

The authors would like to thank all the people that developed the MAX idea into a rich family of different products in different ecosystem, and especially Miller Puckette.

They also thank Ircam for saving jMax from the forgotten software cemetery by publishing its code under a free software license.

Finally, the first author want to thank his old colleagues and friends François and Enzo for accepting to embark on this new adventure.

References

- [1] Di Giugno G. and J. Kott, 1981 : *Présentation du du système 4X Processeur numérique de signal en temps réel*, Rapport IRCAM 32(81), 1981.
- [2] Puckette, M. 1988.: *The Patcher*. Proceedings of the 1988 International Computer Music Conference, San Francisco.
- [3] De Cecco M., E. Lindemann, Puckette M., 1991. *The IRCAM Musical Workstation*. Proceedings of the USENIX 91 Conference, Nashville.
- [4] De Cecco M., Lindemann E., Puckette M., 1992. *The IRCAM Musical Workstation Prototyping Environment*. Proceedings of the CHI'92 Conference, Monterey.
- [5] Déchelle F., De Cecco M., Puckette M., Zicarelli D., 1994. *The Ircam Real-Time Platform : evolution and perspectives*. Proceedings of the 1994 International Computer Music Conference.
- [6] Déchelle F., and De Cecco M., 1995. *The Ircam Real-Time Platform And Applications*. Proceedings of the 1995 International Computer Music Conference.
- [7] Déchelle F., De Cecco M., Maggi E., and Schnell N., 1996. *New DSP applications on FTS*. Proceedings of the 1996 International Computer Music Conference.
- [8] Déchelle F., Borghesi R., De Cecco M., Maggi E., Rovan B., Schnell N. 1998. *Latest evolutions of the jMax real-time engine: typing, scoping, threading, compiling*. Proceedings of the 1998 International Computer Music Conference.
- [9] Déchelle, F., Borghesi R., De Cecco M., Maggi E., Rovan B., Schnell N. 1998. *jMax: a new JAVA-based editing and control system for real-time applications*. Proceedings of the 1998 International Computer Music Conference.
- [10] Déchelle, F., De Cecco M., Maggi E., Schnell N., 1999. *jMax recent developments*. Proceedings of the 1999 International Computer Music Conference.
- [11] Déchelle, F., Borghesi R., De Cecco M., Maggi E., Rovan E., Schnell N., 1999. *jMax: an advanced environment for real-time musical applications*. Computer Music Journal, octobre 1999.